REMARKS:

This response is being submitted in response to the final Office Action mailed on March 21, 2005. Claims 1-16 stand rejected. Claims 1-16 are pending in the application.

Claim Rejections 35 USC §103

The Examiner has rejected Claims 1-16 under 35 U.S.C. 103(a) as being unpatentable over Programming for VisualAge For Java Version 2 by John Akerley (hereinafter referred to as Akerley). Applicant respectfully traverses this rejection of the claims.

There are numerous differences between the claimed invention and that taught in the Akerly reference that, while in the claims themselves, do not appear to have been given their due weight by the examiner. In particular, Applicant notes that the examiner is incorrectly equating the resource bundles of the Akerly reference with the message catalogs or external files of the claimed invention. They are not equivalent and the distinction is important to understanding the intended scope of Applicant's claims.

The ResourceBundles files from Java are still Java source code files that also contain additional Java source code and use the Java source code syntax. As stated at page three, lines 10-12, of Applicant's specification, "As mentioned, the Java programming language does not store display strings in message catalogs but instead stores language-sensitive areas in a source code data structure referred to as a "ListResourceBundle"." The message catalog or external file of the present invention, conversely, is considered a simple data file. The message catalog syntax does not contain any source code, such as C or C++ programming code. This is reflected in industry practice in which translation vendors always charge less to translate data files instead of Java ResourceBundles files. A reference to page two, lines 2-4 of Applicant's specification makes this clear: "These message catalogs are not source code but are text files, designed for localizability, that allow easy translation of texts into native languages...."

This distinction is important to a proper understanding of the claimed invention. A short discussion of the operation of the claimed invention versus the Akerly reference may be helpful in further illustrating this.

Referring first to the presently claimed invention, the external files we generate are described as message catalog files, which, as described above, are not equivalent to the ResourceBundles files from Java. The ListResourceBundle data structures generated in the claims are generated from the one or more marked localizable strings stored in the external file. In the Akerly reference, the ResourceBundle files are generated directly from the Java source code, as described in connection with Akerly's VisualEdge tool. In the claimed invention, one or more ListResourceBundle data structures are generated from what is stored in the external text file. Storing the marked localizable strings into the external text file is therefore an extra intermediate step, but one that offers important advantages. This approach allows the contents of the ResourceBundles and the changes made to the Java source code "in sync", automatically-precisely because we use these external text files (or message catalogs – see Claim 3). If the source code developer adds or modifies or deletes a localizable string from the Java source code, the present invention can effectively track those changes and merge them into the ResourceBundles files that are eventually generated, again through the use of the external files as intermediate files.

Consider the following, illustrative example. When using Akerly's VisualEdge, the java source code developers mark the localizable string in the java source code directly (versus Claim 1 in which the marked strings are stored in an external text file). Next, after executing a handful of commands available from the VisualEdge tool, VisualEdge converts the marked localizable strings with straightforward java getString() function calls. VisualEdge always generates a call to Java's getString(), which means that it attempts to fetch the localizable string from a ResourceBundle. This has problems, however. What if the marked localizable string did not exist in the localized ResourceBundle because that string was either newly/recently added or modified and not yet translated. If getString() is called in the manner taught by VisualEdge, java will generate an "exception" or error condition. VisualEdge will not automatically generate the original/default English string, for example, as does the present invention. Akerly's document, pages 300-326, does not explain anything about automatically

defaulting to an default string, such as English, in the event that the marked localizable string does not exist (yet) in the localized ResourceBundle.

The present invention, conversely, uses a special function call, described at pages 18-20 of the present application, that knows when to retrieve a localizable string from the localized ResourceBundles and when to default to English if the string is not yet translated (see Claim 5, for instance).

Returning again to Akerly's VisualEdge, when it generates a ResourceBundle, it contains the latest changes that were marked up in the java source code by the java source code developers, being a Java source code file. This means that the localized version of the ResourceBundles is now out-of-date and out-of-sync if the translations started *before* the ResouceBundles were considered "DONE" (or final). With the present invention, this is not an issue because of the use of the external files as intermediate files that can keep the ResourceBundles (the original (English) version and the localized versions) in sync.

This is important for several reasons. First, the present invention allows java source code developers to have flexibility to change the ResourceBundle contents, both its keys and values, without those changes impacting the localized versions of the ResourceBundles. Second, it is important to keep the localized version of the ResourceBundles in-sync with the potentially daily changes that take place in the java source code, so that at any point in time, a pre-released product can be tested, such as for quality purposes, with *partial* translations.

Both of these are possible due to the use of the external files as intermediate files in the present invention. A localized ResourceBundle file is generated from an external file, as recited in the claims. That external file contains a subset of the localizable strings, which may be partially translated. The method knows how to automatically generate a localized version of the ResourceBundle that contains the translated strings, if they exist, and will default to the default language for any new or modified strings that have not yet been translated. There is no need for the translators to do a merge between the original language ResourceBundle and the partially translated ResourceBundle. With Akerly's VisualEdge, the translators will need to do a merge between the translators' previous, localized version of the ResourceBundles with the latest changes introduced by the new version of the

ResourceBundles. Thus, using the approach of the claimed invention can save weeks or more from the product development cycle.

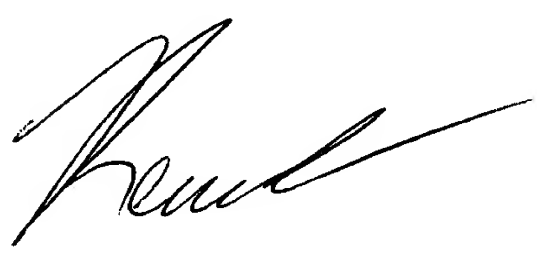
Moreover, consider what happens after a product is officially released and patches are generated to fix defects in the released product. Patches potentially add, delete, or modify the marked localizable strings in the java source code. As such, the mere introduction of such patches can break the translated/localized version of the ResourceBundles if they are not kept in sync with the patched changes in VisualEdge. With the present claimed invention, however, the patched product will still work with the previously, localized version of the ResourceBundle. The default language will be generated for any new/modified/deleted localizable strings until the localized ResourceBundle is also updated to be in sync with the patches. This is not possible with VisualEdge without also updating the localized version of the ResourceBundles with the patched code-a time-consuming and expensive requirement.

In summary it can be seen that the use of an extra step, the storing of marked localizable strings into an external text file, as recited in the claims, allows for a powerful de-coupling approach. Because an intermediate file is introduced that is decoupled from what the source code generates, coding inconsistencies can be eliminated. With the claimed invention, code engineers can make changes to the source code and add/modify/delete localizable strings at any time even after translations have started. This is possible because the translators receive the intermediate files (external or message catalog files) to translate, not the resource bundles. By sending translators simple, message catalog formatted files, the translators and code engineers can operate independently. With other approaches, including VisualEdge, the translators receive the resource bundles themselves. The resource bundles cannot be changed without introducing a synchronization problem between the latest version that the code writers are writing versus the ones that the translators have received. As a consequence, the localization costs can be quite expensive.

Although additional arguments could be made for the patentability of the claims, including claims 2-16, such arguments are believed unnecessary in view of the above discussion of the patentability of claim 1. The undersigned wishes to make it clear that not making such arguments at this time should not be construed as a concession or admission to any statement in the Office Action.

Please contact the undersigned if there are any questions regarding this response or application.

Respectfully submitted,



Renee' Michelle Leveque
Registration No. 36,193
Leveque IP Law, P.C.
221 East Church Street
Frederick, MD 21701
Phone (301) 668-3073
Fax (301) 668-3074

Dated: June 21, 2005